# Spring Boot In 3 Weeks

Day 1: Fundamentals
Day 2: Persistence

# Contact Info


Ken Kousen

Kousen IT, Inc.

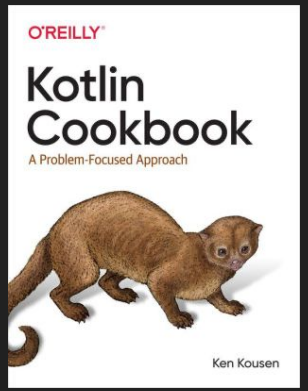
ken.kousen@kousenit.com

http://www.kousenit.com

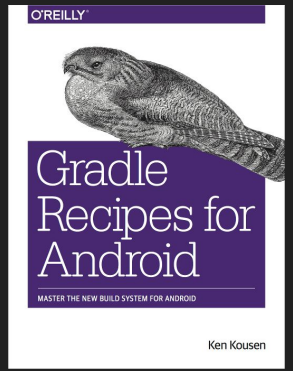
http://kousenit.org (blog)

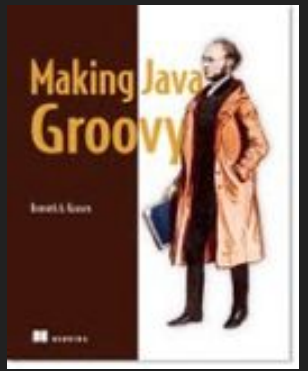
@kenkousen (twitter)

https://kenkousen.substack.com (newsletter)





# New Book

## Help Your Boss Help You

https://pragprog.com/titles/kkmanage/help-your-boss-help-you/

# Spring

Project infrastructure

# Spring

- Lifecycle management of "beans"
  - Any POJO with getters/setters

# Spring

Provides "services"

transactions, security, persistence, …

# Spring

- Library of beans available
  - transaction managers
  - rest clients
  - DB connection pools
  - testing mechanisms

# Spring

Need "metadata"

Tells Spring what to instantiate and configure

XML → old style

Annotations → used for standard components

JavaConfig → used for user-supplied beans

All still supported

# Spring

## Application Context

- Collection of managed beans
- the "lightweight" Spring container

# Spring Boot

- Easy creation and configuration for Spring apps
  - Many "starters"
  - Gradle or Maven based
- Automatic configuration based on classpath
  - If you add JDBC driver, it adds DataSource bean

# Spring Initializr

Website for creating new Spring (Boot) apps

http://start.spring.io

Incorporated into major IDEs

Select features you want

Download zip containing build file

# Spring Boot

Application with main method created automatically

Annotated with `@SpringBootApplication`

Gradle or Maven build produces executable jar in build/libs folder

```
$ java -jar appname.jar
```

Or use gradle task `bootRun`

# Spring MVC

Annotation based MVC framework

`@Controller` → controllers

`@GetMapping` → annotations for HTTP methods

`@RequestParam` and more for model parameters

`Model` interface → map for carrying data from one resource to another

# Rest Client

Spring includes a class called `RestTemplate`

- Access RESTful web services
- Set HTTP methods, headers, query string, templates
- Use `RestTemplateBuilder` to create one
- Use content negotiation to return JSON or XML
- Convenient `getForObject(url, class)` method

Newer reactive client: WebClient

# Logging

Spring libraries include SLF4J automatically

- Use `LoggerFactory.getLogger(... class name ...)`
- Returns an `org.slf4j.Logger` instance
- Invoke logging methods as usual

# Dependency Injection

- Spring adds dependencies on request
  - Annotate field, or setter, or constructor
  - `@Autowired` → autowiring by type
  - `@Resource` (from Java EE) → autowiring by (bean) name, then by type if necessary

# Testing

Spring tests automatically include special JUnit 5 extension

- `@ExtendWith(SpringExtension.class)`
- Annotate test class with `@SpringBootTest`
- Annotate tests with `@Test`
- Use normal asserts as usual

# Unit Testing

Instantiate class and invoke methods

Dependencies can be mocked → Mockito is already included

Fast, but least realistic

# Integration Testing

Special annotations for web integration tests

Uses Spring, but not an actual server

`@WebMvcTest(... controller class ...)`

`MockMvc` package

`MockMvcRequestBuilders`

`MockMvcRequestMatchers`

# Functional Testing

Run on an actual test server

```
@SpringBootTest(webEnvironment = RANDOM)
```

- Spring chooses random port
- Deploys app
- Runs tests
- Shuts down server

Most realistic, but potentially slow

# Parsing JSON

- Several options, but one is the `Jackson JSON 2` library
  - Create classes that map to JSON response
  - `restTemplate.getForObject(url, ... your class ...)`
- Maps JSON to Java objects

# Component Scan

Spring detects annotated classes in the expected folders

`@Component` → Spring bean

`@Controller`, `@Service`, `@Repository` → based on `@Component`

# Application properties

Two options for file name

Default folder is `src/main/resources`

- `application.properties` → standard Java properties file
- `application.yml` → YAML format

# Summary for Week 1

Spring:

- Dependency injection
- Provides services
- Includes large API

Spring Boot:

- Used to create a new Spring app
- Auto-configures many beans

Great for web apps, restful web services, and more

# Day 2: Persistence

JdbcTemplate → Pass SQL to DB

JPA → Use Java Persistence API

Spring Data JPA → Generate your entire DAO layer

# Persistence

- Spring provides `JdbcTemplate`
  - Easy to access and use relational databases
  - Best if you already have the SQL you want to use

# Persistence

More conventions:

- Two standard files in `src/main/resources`
- `schema.sql` → create test database
- `data.sql` → populate test database

Both executed on startup, using DB connection pool

# JdbcTemplate

Standard practice:

- Create DAO interface and implementation class
- Autowire `DataSource` into constructor
- Instantiate `JdbcTemplate` from `DataSource`

Spring Boot lets you autowire the `JdbcTemplate` directly

# JdbcTemplate

Use `queryForObject` to map DB row to Java class

(`query` method does the same for all rows)

In Java 7, uses inner class that implements `RowMapper<MyClass>`

In Java 8, can use lambda expression

# H2 Database

- Add the H2 dependency
  - `runtime('com.h2database:h2')`
  - Automatically adds DataSource for it

If you add the web starter and the dev-tools dependency,

H2 console: http://localhost:8080/h2-console

DB URL in console of the form **`jdbc:h2:mem:<generated>`**

# SimpleJdbcInsert

- Specify table name and generated key columns
- Create a `SqlParameterSource` or a `Map`
- Run `executeAndReturnKey(parameters)`

# Transactions

Spring transactions configured with `@Transactional`

Spring uses `TransactionManager` to talk to resource

usually a relational DB, but other options available

# @Transactional

Each method wrapped in a REQUIRED tx by default

Propagation levels:

REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED

In tests, transactions in test methods roll back by default

Can configure isolation levels:

READ_UNCOMMITTED, READ_COMMITTED,

REPEATABLE_READ, SERIALIZABLE

# JPA

Java Persistence API

- Uses a "provider" → Hibernate most common

Annotate entity classes

- `@Entity, @Table, @Column, @Id, @GeneratedValue`

use in Spring `@Repository` → exception translation

- `@PersistenceContext` → `EntityManager`

# Spring Data

Large, powerful API

Create interface that extends a given one

`CrudRepository, PagingAndSortingRepository`

We'll use `JpaRepository<class, serializable>`

Add your own finder method declarations

All SQL generated automatically

# Summary for Week 2

Persistence:

- JdbcTemplate, SimpleJdbcInsert
- `@PersistenceContext` for JPA
- Spring Data JPA → generate entire DAO layer

Transactions:

- `@Transactional` annotation
- Can set isolation level and propagation levels